ECE 253a	Handout #14	11/11/10

LEMPEL-ZIV CODING

Flavors of LZ coding:
Most adaptive-dictionary-based techniques have their roots in two landmark papers by Jacob Ziv and Abraham Lempel in 1977 and 1978.
In the LZ77 paper, the approach was to encode the next string by using the recently encoded past. Within the search buffer, the encoder would search for the longest match to the next string. The encoder would encode the position of the longest match in the search window (called the offset) and the length of the match. Several methods have been proposed for handling the situation where the next symbol to be encoded does not occur at all in the recent past. The offset and the length can themselves be variable-length coded or fixed-length coded.
The type discussed in this handout is from the LZ78 family, in which an explicit dictionary is created identically at encoder and decoder.

- An input sequence is recursively parsed into nonoverlapping blocks of variable size while constructing a dictionary of blocks seen so far
- The dictionary is initialized with the available single symbols, 0 and 1 for a binary source
- Each successive block in the parsing (encoding) is chosen to be the longest word w that has appeared in the dictionary
- If the next symbol after w is a, the word wa formed by concatenating w and a is not in the dictionary. We add it to the dictionary and go on, with a becoming the first symbol in the next block to be coded.

Example of encoding: We wish to encode the following binary string: 01100110010110000100110
We begin with the table

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001

Step 1: Longest string in Table = 0
Output entry # = 0
Add to table: 01
pointer = 2. The pointer indicates which symbol will start the next block to be encoded. New table is:

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001
01	2	00000010

Step 2: Longest string in Table = 1
Output entry #= 1
Add to table: 11
pointer = 3.

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001
01	2	00000010
11	3	00000011

Step 3: Longest string in Table = 1
Output entry # = 1
Add to table: 10
pointer = 4.

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001
01	2	00000010
11	3	00000011
10	4	00000100

Step 4: Longest string in Table = 0
Output entry # = 0
Add to table: 00
pointer = 5.

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001
01	2	00000010
11	3	00000011
10	4	00000100
00	5	00000101

Step 5: Longest string in Table = 01
Output entry # = 2
Add to table: 011
pointer = 7.

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001
01	2	00000010
11	3	00000011
10	4	00000100
00	5	00000101
011	6	00000110

Step 6: Longest string in Table = 10
Output entry # = 4
Add to table: 100
pointer = 9.

Input string	Entry #	Dictionary Index
0	0	00000000
1	1	00000001
01	2	00000010
11	3	00000011
10	4	00000100
00	5	00000101
011	6	00000110
100	7	00000111

Step 7: Longest string in Table = 01
Output entry # = 2
Add to table: 010
pointer = 11. And so forth.

Decoder operation:

The decoder starts with the same table and hence immediately decodes the first symbol as a 0 and the second as 1. The decoder recognizes the new string 01 and adds it to its table as entry # 2. The next symbol is a 1, which means that an input of 1 was seen by the encoder. The decoder sees the string 11 terminating with the current symbol and adds it to its table as entry # 3. On seeing the fourth symbol 0, the decoder again knows that the encoder saw a 0, and it recognizes the sequence 10 terminating with the current symbol as a new one for its table with index 4. etc.

A fix to this basic LZ algorithm:

The algorithm as described can have a problem, in that the encoder can add an entry to its dictionary and then use it immediately, before the decoder seemingly knows what it is. But there is a way the decoder can figure out this type of case. Here is an example of the problem with a ternary alphabet {0,a,b}. Initial table:

Input string	Entry #
0	0
a	1
b	2

The input sequence is a000ba0a...
The encoder will take the following actions:

Step	Send	New Entry	Entry #
1	1 (for a)	a0	3
2	0 (for 0)	00	4
3	4 (for 00)	00b	5
4	2 (for b)	b0	6
5	3 (for a0)	a0a	7

The decoder will receive the index for entry 1 and will decode the “a”. Then it will receive the index for entry 0 and decode a “0” and additionally will be able to add the word a0 to its dictionary with index 3. But then the decoder receives the index for entry 4! And it doesn’t yet have entry 4 in its dictionary! But the decoder is able to figure this out. It knows that entry 4 (the current block) must have the form x? where x is the previously encoded block, and ? is a single letter. Since the previously encoded block was a 0, entry 4 must be of the form 0?. But the decoder also knows that the last letter of the last block (the last letter is the ?) is the first letter of the current block (the current block is 0?) so ? is 0. Thus entry 4 is 00, and the decoder can go on.